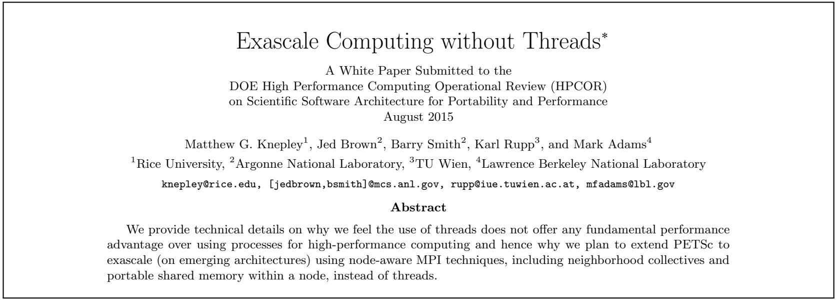
Exascale Computing Without Templates

Karl Rupp, Richard Mills, Barry Smith, Matthew Knepley, Jed Brown

Argonne National Laboratory

DOE COE Performance Portability Meeting, Denver
August 22-24, 2017

Providing Context



Exascale Computing without Threads*

A White Paper Submitted to the
DOE High Performance Computing Operational Review (HPCOR)
on Scientific Software Architecture for Portability and Performance
August 2015

Matthew G. Knepley[1], Jed Brown[2], Barry Smith[2], Karl Rupp[3], and Mark Adams[4]

[1]Rice University, [2]Argonne National Laboratory, [3]TU Wien, [4]Lawrence Berkeley National Laboratory

knepley@rice.edu, [jedbrown,bsmith]@mcs.anl.gov, rupp@iue.tuwien.ac.at, mfadams@lbl.gov

Abstract

We provide technical details on why we feel the use of threads does not offer any fundamental performance advantage over using processes for high-performance computing and hence why we plan to extend PETSc to exascale (on emerging architectures) using node-aware MPI techniques, including neighborhood collectives and portable shared memory within a node, instead of threads.

https://www.orau.gov/hpcor2015/whitepapers/Exascale_Computing_without_Threads-Barry_Smith.pdf

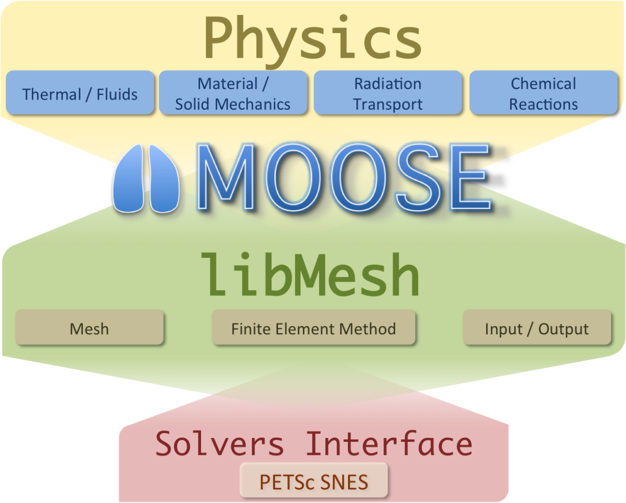
PETSc Developers Care About Recent Developments

After careful evaluation: Favor MPI 3.0 (and later) over threads

Find the best long-term solutions for our users

Consider best solutions for large-scale applications, not just toy-apps

Providing Context



http://mooseframework.org/static/media/wiki/images/229/b61cdbc1e8be71dae37adc31a688d209/moose-arch.png

Providing Context

Our Attempts in C++ Library Development

- Sieve: Several years of C++ mesh management attempts in PETSc
- ViennaGrid 2.x: Heavily templated C++ mesh management library
- ViennaCL: Dense and sparse linear algebra and solvers for multi- and many-core architectures

Providing Context

Our Attempts in C++ Library Development

Sieve: Several years of C++ mesh management attempts in PETSc

ViennaGrid 2.x: Heavily templated C++ mesh management library

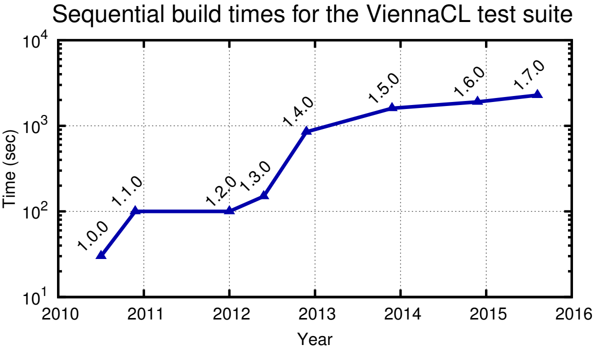
ViennaCL: Dense and sparse linear algebra and solvers for multi- and many-core architectures

Aftermath

Sieve: Replaced by DMPlex (written in C)

ViennaGrid: Version 3.0 provides C-ABI

ViennaCL: Rewrite in C likely



Disadvantages of C++ Templates

Static Dispatch

Architecture-specific information only available at run time

“Change code and recompile” not acceptable advice

Disadvantages of C++ Templates

Static Dispatch

Architecture-specific information only available at run time

“Change code and recompile” not acceptable advice

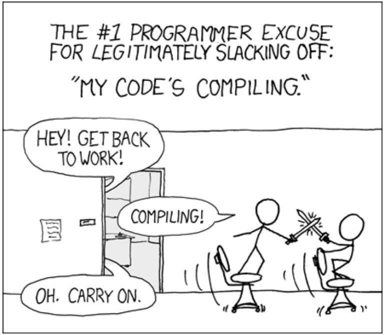
Dealing with Compilation Errors

Type names pollute compiler output

Replicated across interfaces

CRTP may result in type length explosion

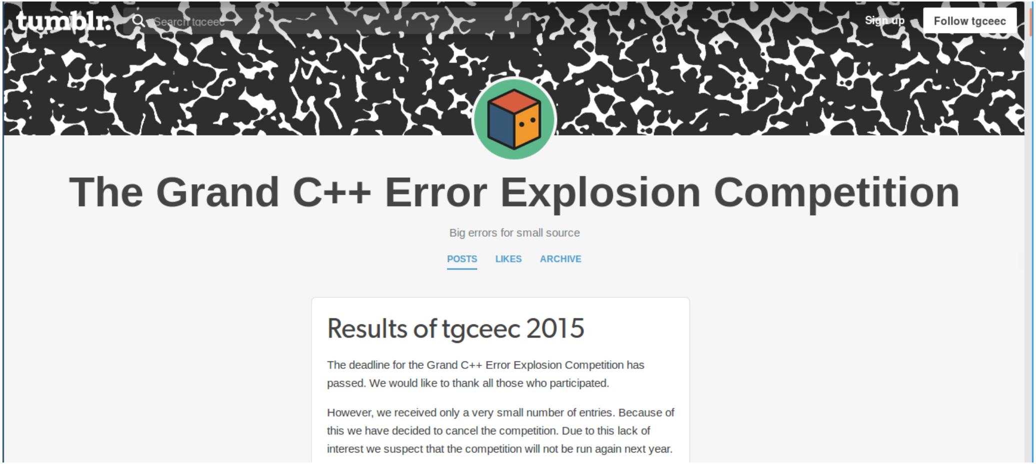
Default arguments become visible



https://xkcd.com/303/

Type	Length
`std::vector<int>`	38
`std::vector<std::vector<int> >`	109
`std::vector<std::vector<std::vector<int> > >`	251
`std::vector<std::vector<std::vector<std::vector<int> > > >`	539

Disadvantages of C++ Templates



The Grand C++ Error Explosion Competition

Big errors for small source

Results of tgceec 2015

The deadline for the Grand C++ Error Explosion Competition has passed. We would like to thank all those who participated.

However, we received only a very small number of entries. Because of this we have decided to cancel the competition. Due to this lack of interest we suspect that the competition will not be run again next year.

https://tgceec.tumblr.com/

Disadvantages of C++ Templates

Scope Limitations

- Template metaprogramming lacks state
- Optimizations across multiple code lines difficult or impossible

Disadvantages of C++ Templates

Scope Limitations

Template metaprogramming lacks state

Optimizations across multiple code lines difficult or impossible

Example

Consider vector updates in pipelined CG method:

$$\begin{aligned} x_i &\leftarrow x_{i-1} + \alpha p_{i-1} \\ r_i &\leftarrow r_{i-1} - \alpha y_i \\ p_i &\leftarrow r_i + \beta p_{i-1} \end{aligned}$$

Reuse of p_{i-1} and r_{i-1} easy with for-loops, but hard with expression templates

Disadvantages of C++ Templates

Complicates Debugging

- Stack traces get longer names and deeper
- Setting good breakpoints may become harder

Disadvantages of C++ Templates

Complicates Debugging

Stack traces get longer names and deeper

Setting good breakpoints may become harder

Lack of a Stable ABI

Object files from different compilers generally incompatible

Name mangling makes use outside C++ land almost impossible

Disadvantages of C++ Templates

Complicates Debugging

Stack traces get longer names and deeper

Setting good breakpoints may become harder

Lack of a Stable ABI

Object files from different compilers generally incompatible

Name mangling makes use outside C++ land almost impossible

High Entry Bar

Number of potential contributors inversely proportional to code sophistication

Domain scientists have limited resources for C++ templates

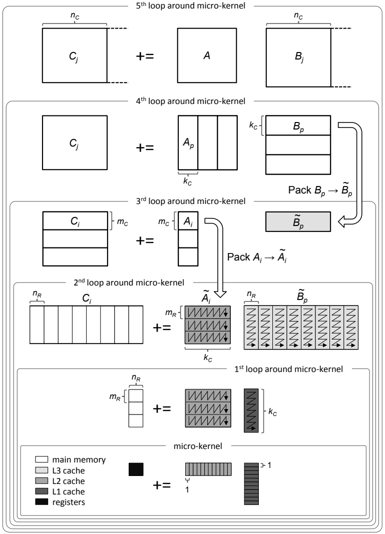
A Path Forward

Manage Complexity

Good interface design

Refactor code when needed

Hand-optimize small kernels only (cf. BLIS methodology)



[F. Van Zee, T. Smith, ACM TOMS 2017]

A Path Forward

Manage Complexity

Good interface design

Refactor code when needed

Hand-optimize small kernels only (cf. BLIS methodology)

Development Implications

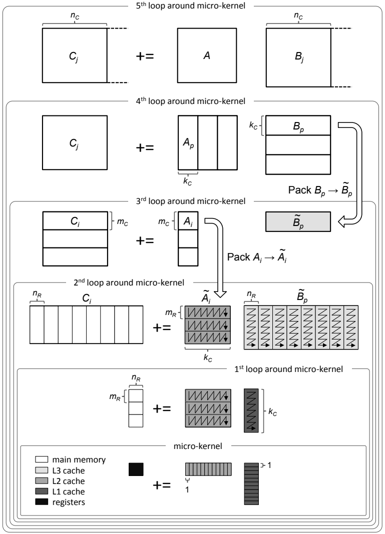
Adopt professional software development practices

Develop, maintain, and evolve different datastructures ...

... and code paths

Use clear and easy-to-understand datastructures

Fallacy: “Writing” an application only once in its final form



[F. Van Zee, T. Smith, ACM TOMS 2017]

A Path Forward

Spending Development Resources

Reuse existing libraries — reinventing the wheel is not productive!

Focus on domain- and application-specific aspects

Obtain expertise and resources for continuous code evolution

A Path Forward

Spending Development Resources

Reuse existing libraries — reinventing the wheel is not productive!

Focus on domain- and application-specific aspects

Obtain expertise and resources for continuous code evolution

Required Incentives

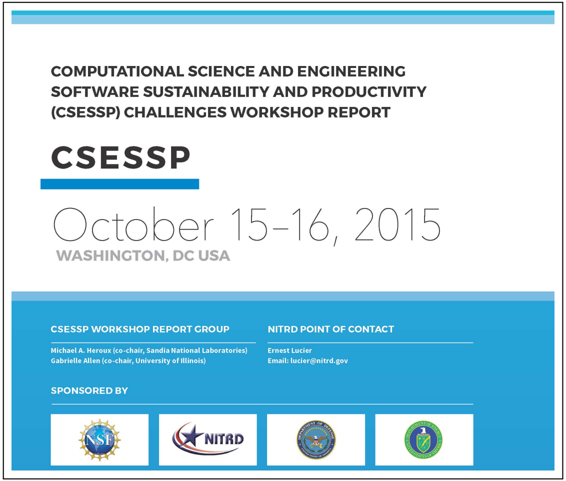
Reward contributions to existing projects

Pair research funding with software development funding

Establish software development career tracks

A Path Forward

Is Performance Portability Just a Software Productivity Aspect?



COMPUTATIONAL SCIENCE AND ENGINEERING
SOFTWARE SUSTAINABILITY AND PRODUCTIVITY
(CSESSP) CHALLENGES WORKSHOP REPORT

CSESSP

October 15–16, 2015
WASHINGTON, DC USA

CSESSP WORKSHOP REPORT GROUP

Michael A. Heroux (co-chair, Sandia National Laboratories)
Gabrielle Allen (co-chair, University of Illinois)

NITRD POINT OF CONTACT

Ernest Lucier
Email: lucier@nitrd.gov

SPONSORED BY

https://www.nitrd.gov/PUBS/CSESSPWorkshopReport.pdf

Summary

Long-Term Problems of Heavy C++ Templates Use

Template metaprogramming is a leaky abstraction

Excessive type names slow down all stages of Compile-Run-Debug-cycle

Templates operate at compile time - architecture ultimately known at run time

A Path Forward

Adopt professional software development practices

Be prepared to develop different datastructures and code paths

Write clear, readable code using simple datastructures

Evolve and refactor datastructures, kernels, and interfaces over time

(cf. software productivity discussions)